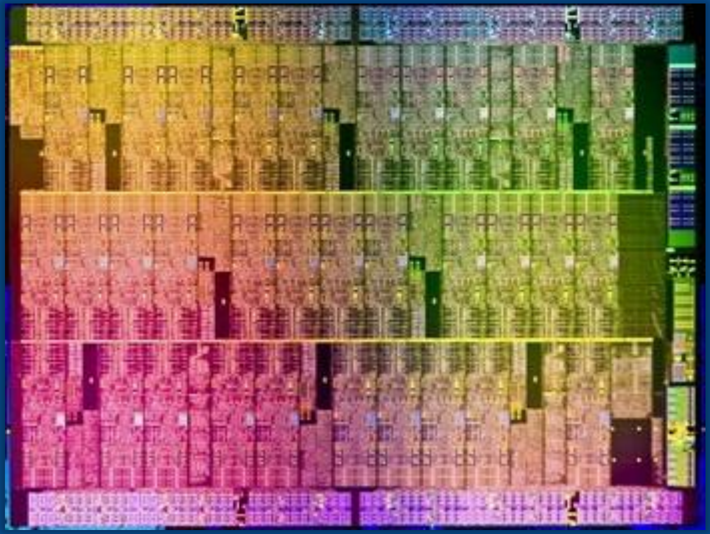
Improving Performance of Weather and Climate Applications on Multi-/Many-core Processors



John Michalakes

Computational Sciences Center
NREL

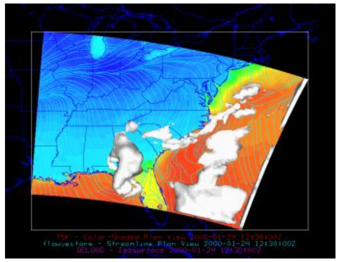
Workshop on programming weather, climate, and earth-system models on heterogeneous multi-core platforms

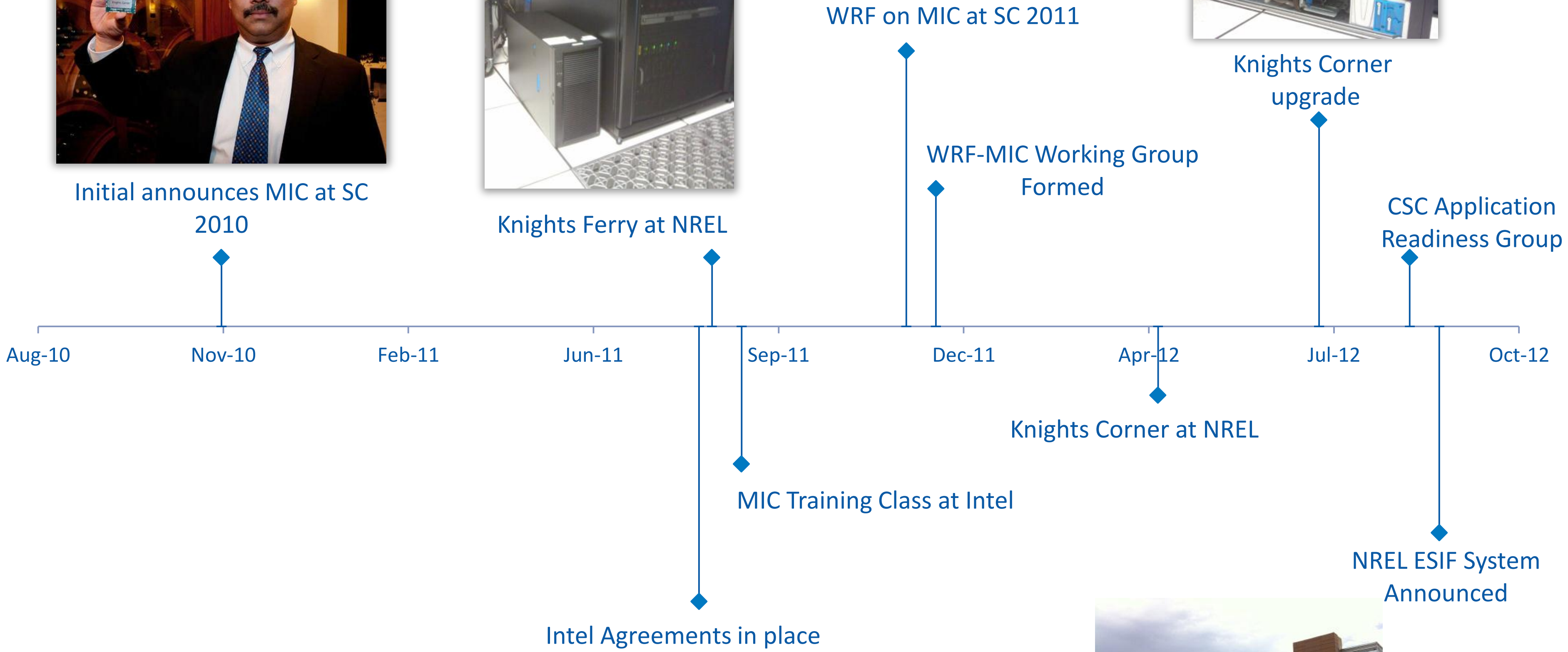
September 13, 2012

Acknowledgements

- Intel Corp.
  - Michael Greenfield
  - Antonio Valles
  - Lawrence Meadows
  - Indraneil Gokhale
  - Alexander Knyazev
  - Mark West
  - Brian Rea
  - Ruchira Sasanka

Intel MIC at NREL




Initial announces MIC at SC 2010
Knights Ferry at NREL
WRF on MIC at SC 2011
WRF-MIC Working Group Formed
Knights Corner upgrade
CSC Application Readiness Group
Aug-10
Nov-10
Feb-11
Jun-11
Sep-11
Dec-11
Apr-12
Jul-12
Oct-12
Knights Corner at NREL
MIC Training Class at Intel
NREL ESIF System Announced
Intel Agreements in place

WRF on MIC

- Community regional weather model developed and supported at NCAR in use at NREL for
  - Wind and solar assessments
  - Wind plant aerodynamics
  - Offshore wind environments
- High performance computing
  - Hybrid parallel
    - MPI parallel using 2-D horizontal domain decomposition
    - Each MPI subdomain further subdivided over OpenMP threads
    - Significant potential for vector parallelization
  - Computational footprint
    - Relatively low arithmetic intensity and limited operand reuse
    - Large working set sizes overflow caches
  - Generally good coarse-grain scaling but relatively poor processor performance as a percentage of peak

Improving WRF on Multi-/Many-core (easy)

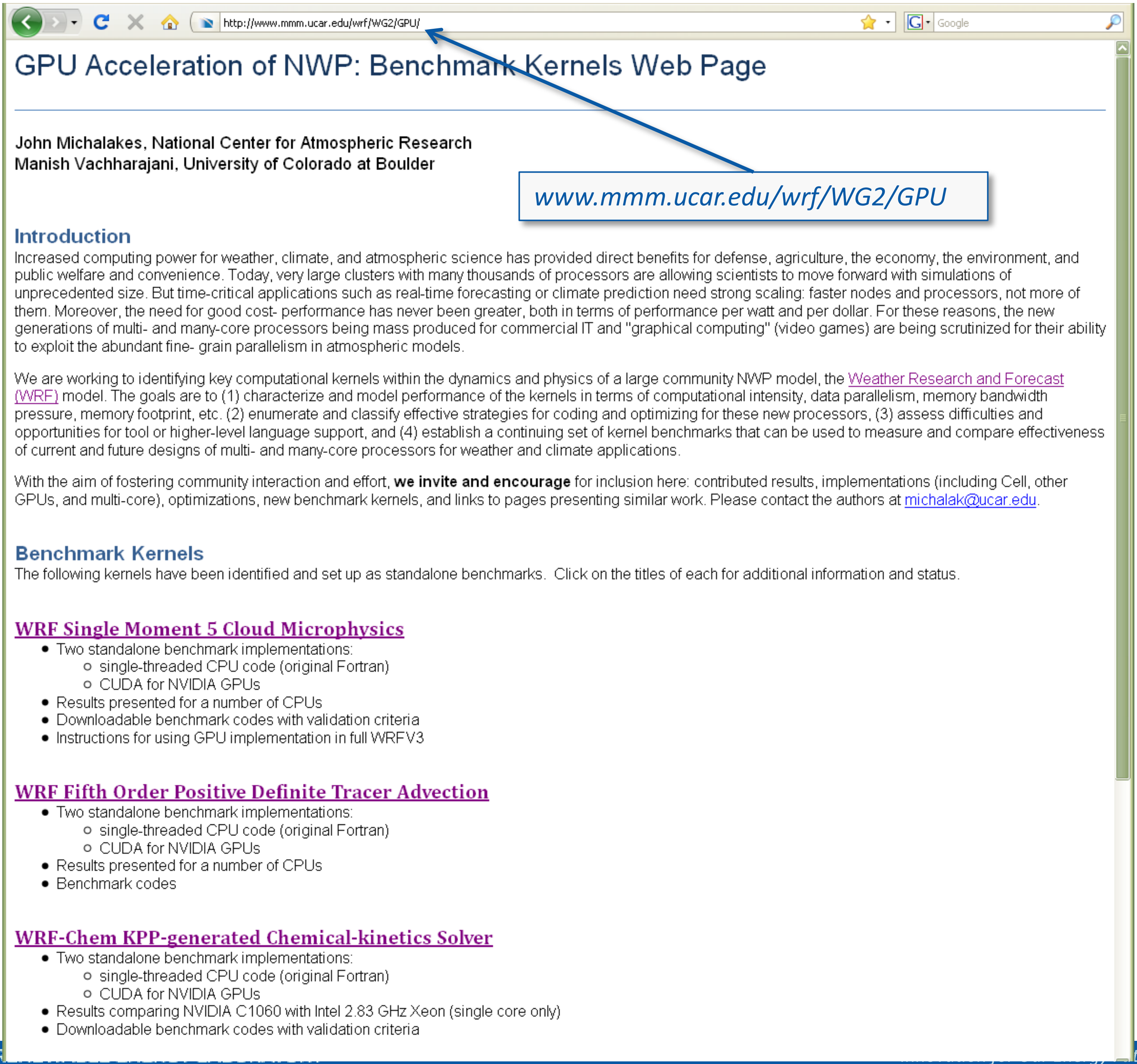
- Options and directives
  - -O3 (-fp-model-precise may also be needed)
  - -align all, -fno-alias
  - -vec-report3, detailed report of what loops vectorized or why not
  - $DEC! SIMD or $DEC! VECTOR ALWAYS
  - DYNAMIC or STATIC clauses for OpenMP PARALLEL DO
- Environment
  - KMP_AFFINITY (scatter is usually good)
  - OMP_STACKSIZE
  - OMP_NUM_THREADS

Improving WRF on Multi-/Many-core

- Moving the MPI and OpenMP transition
  - Need to cover 50+ cores each running 1-4 threads
    - 1 task and all parallelism through threading
    - MPI task-per-core with 1-4 threads for latency hiding
    - Something in between
  - On conventional multicore best performance has been with straight MPI
    - Manages locality
    - Overhead for message passing is sub-optimal for multi-core
      - Wastes space – buffers and duplicated application storage
      - Wastes time and BW – copying in MPI and for message aggregation
      - Wastes memory bandwidth
      - One-sided cache-to-cache models, e.g. PGAS?

Improving WRF on Multi-core (harder)

- Code restructuring
  - Fusing loops (beyond what compiler is able to do)
  - Tiling into statically sized thread-local arrays
    - Helps shrink and align working set for cache and vector friendliness and makes array dimensions available to the compiler
    - Usefully employed with WRF microphysics module
    - Assumes enough work within routine to offset cost of copying into and out of tile-sized arrays
    - Functional fusion can provide even more work and reuse
  - Storage order and loop nesting order
    - WRF arrays and loops are i,k,j with i-innermost/fastest varying
    - Switching to k,i,j in combination with tiling, above, may improve vector performance and cache locality
- Different algorithms (hardest)


http://www.mmm.ucar.edu/wrf/WG2/GPU/

www.mmm.ucar.edu/wrf/WG2/GPU

GPU Acceleration of NWP: Benchmark Kernels Web Page

John Michalakes, National Center for Atmospheric Research
Manish Vachharajani, University of Colorado at Boulder

Introduction

Increased computing power for weather, climate, and atmospheric science has provided direct benefits for defense, agriculture, the economy, the environment, and public welfare and convenience. Today, very large clusters with many thousands of processors are allowing scientists to move forward with simulations of unprecedented size. But time-critical applications such as real-time forecasting or climate prediction need strong scaling: faster nodes and processors, not more of them. Moreover, the need for good cost- performance has never been greater, both in terms of performance per watt and per dollar. For these reasons, the new generations of multi- and many-core processors being mass produced for commercial IT and "graphical computing" (video games) are being scrutinized for their ability to exploit the abundant fine- grain parallelism in atmospheric models.

We are working to identifying key computational kernels within the dynamics and physics of a large community NWP model, the Weather Research and Forecast (WRF) model. The goals are to (1) characterize and model performance of the kernels in terms of computational intensity, data parallelism, memory bandwidth pressure, memory footprint, etc. (2) enumerate and classify effective strategies for coding and optimizing for these new processors, (3) assess difficulties and opportunities for tool or higher-level language support, and (4) establish a continuing set of kernel benchmarks that can be used to measure and compare effectiveness of current and future designs of multi- and many-core processors for weather and climate applications.

With the aim of fostering community interaction and effort, **we invite and encourage** for inclusion here: contributed results, implementations (including Cell, other GPUs, and multi-core), optimizations, new benchmark kernels, and links to pages presenting similar work. Please contact the authors at michalak@ucar.edu.

Benchmark Kernels

The following kernels have been identified and set up as standalone benchmarks. Click on the titles of each for additional information and status.

WRF Single Moment 5 Cloud Microphysics

- Two standalone benchmark implementations:
  - single-threaded CPU code (original Fortran)
  - CUDA for NVIDIA GPUs
- Results presented for a number of CPUs
- Downloadable benchmark codes with validation criteria
- Instructions for using GPU implementation in full WRFV3

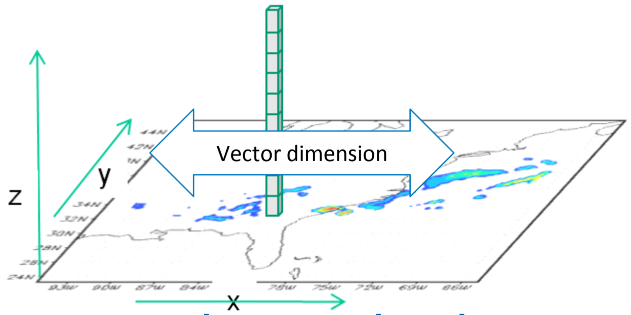
WRF Fifth Order Positive Definite Tracer Advection

- Two standalone benchmark implementations:
  - single-threaded CPU code (original Fortran)
  - CUDA for NVIDIA GPUs
- Results presented for a number of CPUs
- Benchmark codes

WRF-Chem KPP-generated Chemical-kinetics Solver

- Two standalone benchmark implementations:
  - single-threaded CPU code (original Fortran)
  - CUDA for NVIDIA GPUs
- Results comparing NVIDIA C1060 with Intel 2.83 GHz Xeon (single core only)
- Downloadable benchmark codes with validation criteria

Adapting Microphysics to MIC



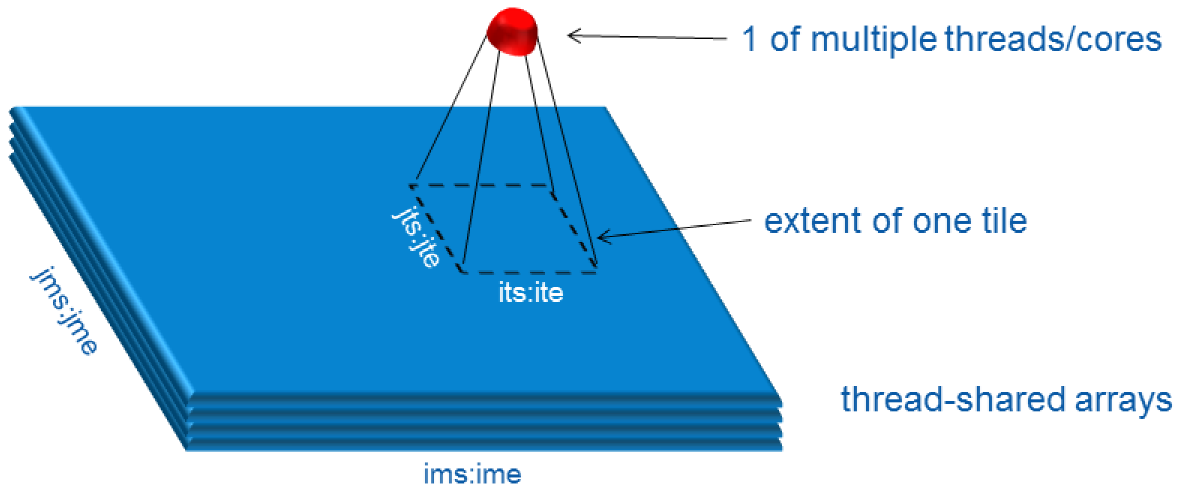
- First attempt
  - Compiled “vanilla” WSM5 kernel with OpenMP and vectorization
  - Applied simple optimization to fuse loops over vertical

    ~8% improvement
- Additional optimization:
  - Convert arrays to thread-local tiles with compile-time-known dimensions
  - Copy into and out of these arrays on each OpenMP thread

Adapting Microphysics to MIC

- First atte
  - Comp
  - Appli
    ~8%
- Addition
  - Conv dimer
  - Copy

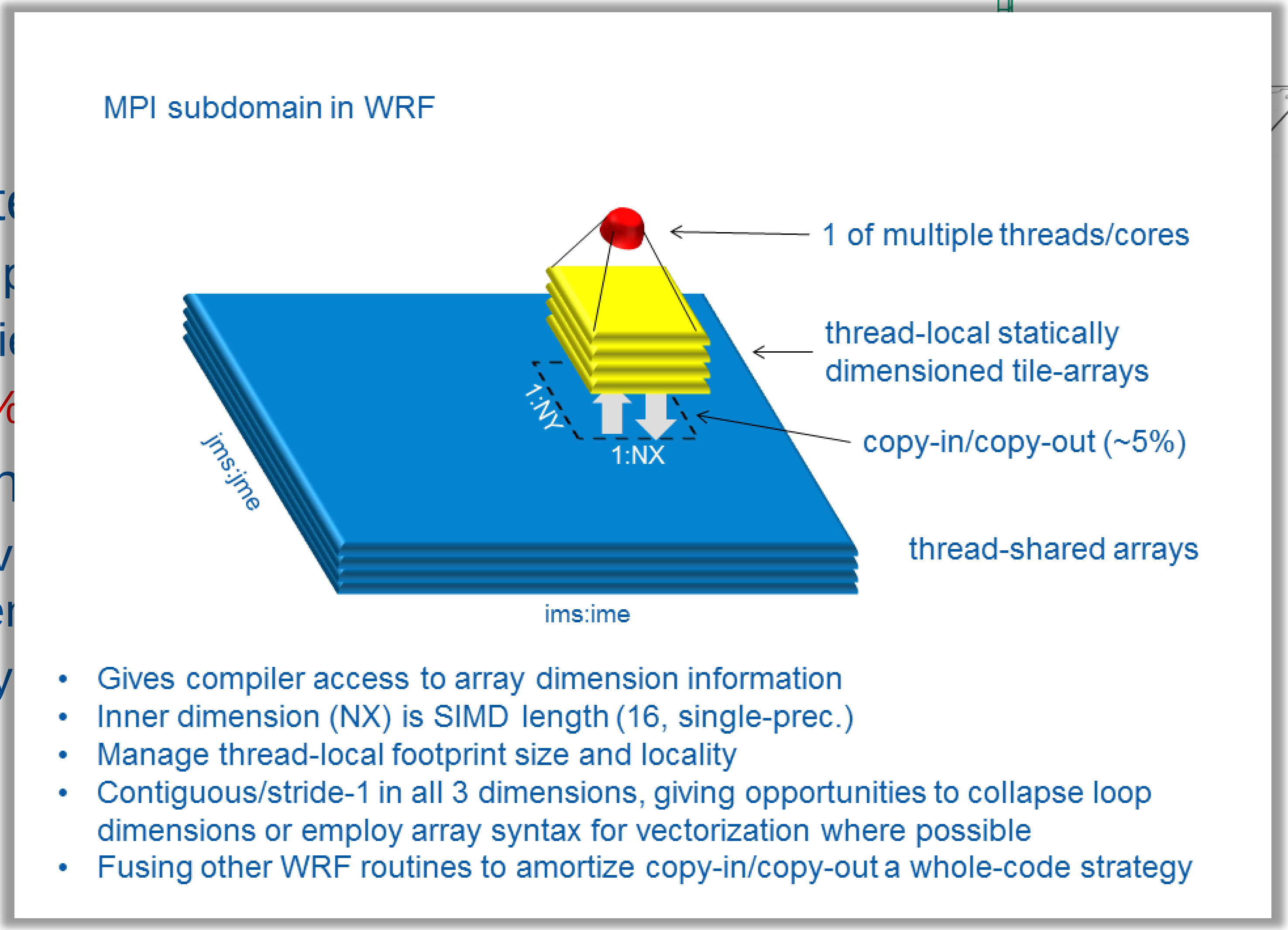
MPI subdomain in WRF



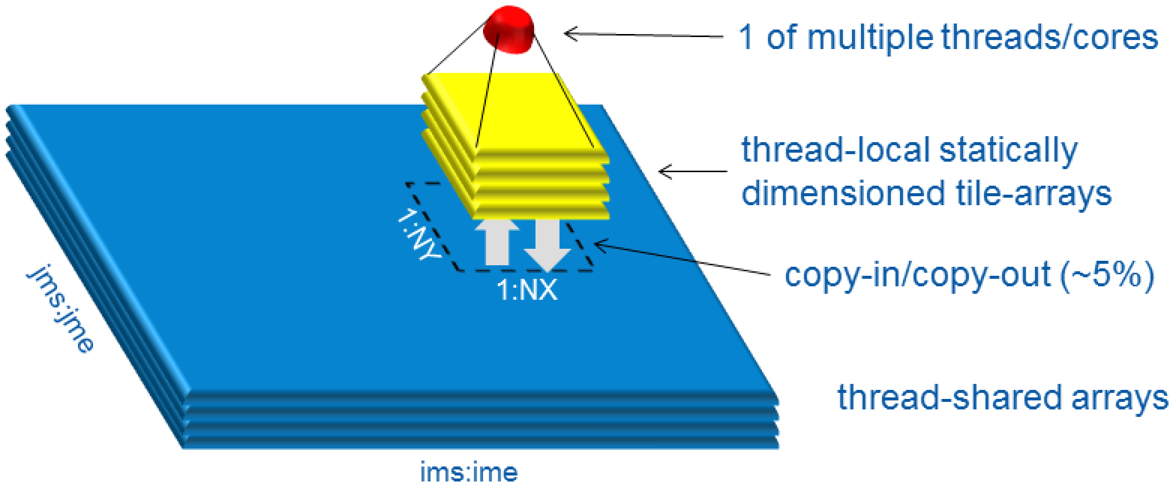
- Dimensions of arrays and tile loop extents known only at run-time
- Stride-1 sequences of operands short and often interrupted
- Difficult to manage thread-local memory footprint

Adapting Microphysics to MIC

- First atte
  - Comp
  - Appli
    ~8%
- Addition
  - Conv dimer
  - Copy

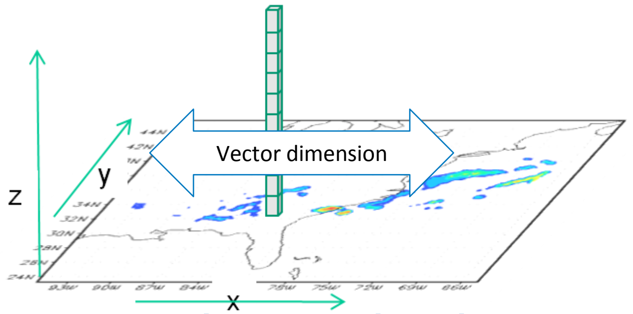


MPI subdomain in WRF


1 of multiple threads/cores
thread-local statically dimensioned tile-arrays
copy-in/copy-out (~5%)
thread-shared arrays
1:NY
1:NX
jms:jme
ims:ime

- Gives compiler access to array dimension information
- Inner dimension (NX) is SIMD length (16, single-prec.)
- Manage thread-local footprint size and locality
- Contiguous/stride-1 in all 3 dimensions, giving opportunities to collapse loop dimensions or employ array syntax for vectorization where possible
- Fusing other WRF routines to amortize copy-in/copy-out a whole-code strategy

Adapting Microphysics to MIC

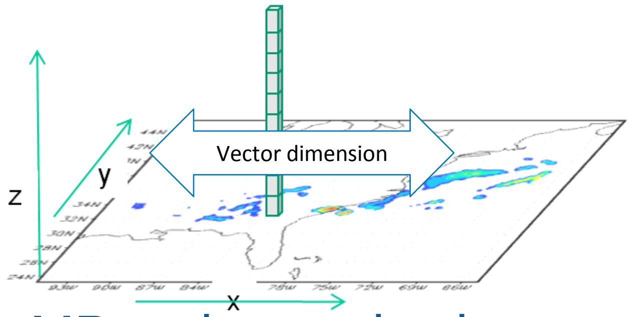


- First attempt
  - Compiled “vanilla” WSM5 kernel with OpenMP and vectorization
  - Applied simple optimization to fuse loops over vertical

    ~8% improvement
- Additional optimization:
  - Convert arrays to thread-local tiles with compile-time-known dimensions
  - Copy into and out of these arrays on each OpenMP thread

    > 2x performance improvement

Adapting Microphysics to MIC



- First attempt
  - Compiled “vanilla” WSM5 kernel with OpenMP and vectorization
  - Applied simple optimization to fuse loops over vertical

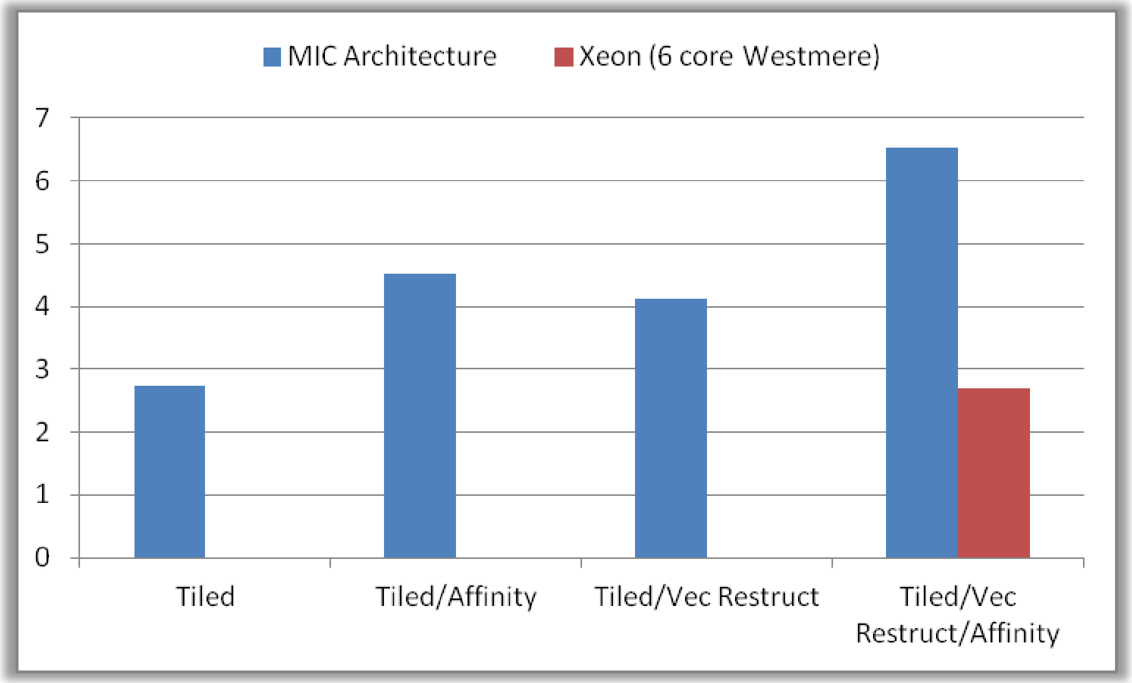
    ~8% improvement
- Additional optimization:
  - Convert arrays to thread-local tiles with compile-time-known dimensions
  - Copy into and out of these arrays on each OpenMP thread

    > 2x performance improvement
- Whole-code performance improvement

  1.11x (theoretically possible: 1.33x)

Effects of optimization

- Performance improvement relative to original Fortran on Knights Ferry for WRF Microphysics Kernel
- Changes for Intel MIC coprocessors improve results on Intel Xeon processors too



Reactive Atmospheric Chemistry Solver

- WRF-Chem RADM/SORGAM configuration (Grell, Peckham)
  - 61 species
  - 159 reactions
  - 28889 cell domain, each cell independent (chem is O(10) cost meteo.)
  - KPP generated Rosenbrock Solver, double precision
- Linford, J.; Michalakes, J.; Vaccharajani, M.; Sandu, A. (2009). "Multi-core Acceleration of Chemical Kinetics for Modeling and Simulation." *Proceedings SC'09*
  - GPU, Cell, and multithreaded multicore implementations
  - Very large working set, very low computational intensity
  - Difference between good and bad performance was ability to fit in cache/local store on Cell BE and Nehalem, but not on GPU
  - MIC cores have L2 caches similar size to Cell BE

Reactive Atmospheric Chemistry Solver

- MIC implementation
  - Collapse and OpenMP thread of triply nested loop over cells of WRF-Chem domain
  - Dynamic thread scheduling
  - Add an inner dimension over cells for each array in Rosenbrock solver with masking to turn off cells as they reach convergence

Reactive Atmospheric Chemistry Solver

- MIC implementation
  - Collapse and OpenMP thread of triply nested loop over cells of WRF-Chem domain
  - Dynamic thread scheduling
  - Add an inner dimension over cells for each array in Rosenbrock solver with masking to turn off cells as they reach convergence

```
W( 53 ) = -a
W( 54 ) = W( 54 ) + a*JVS( 506 )
W( 55 ) = W( 55 ) + a*JVS( 507 )
W( 56 ) = W( 56 ) + a*JVS( 508 )
W( 57 ) = W( 57 ) + a*JVS( 509 )
W( 58 ) = W( 58 ) + a*JVS( 510 )
W( 59 ) = W( 59 ) + a*JVS( 511 )
```

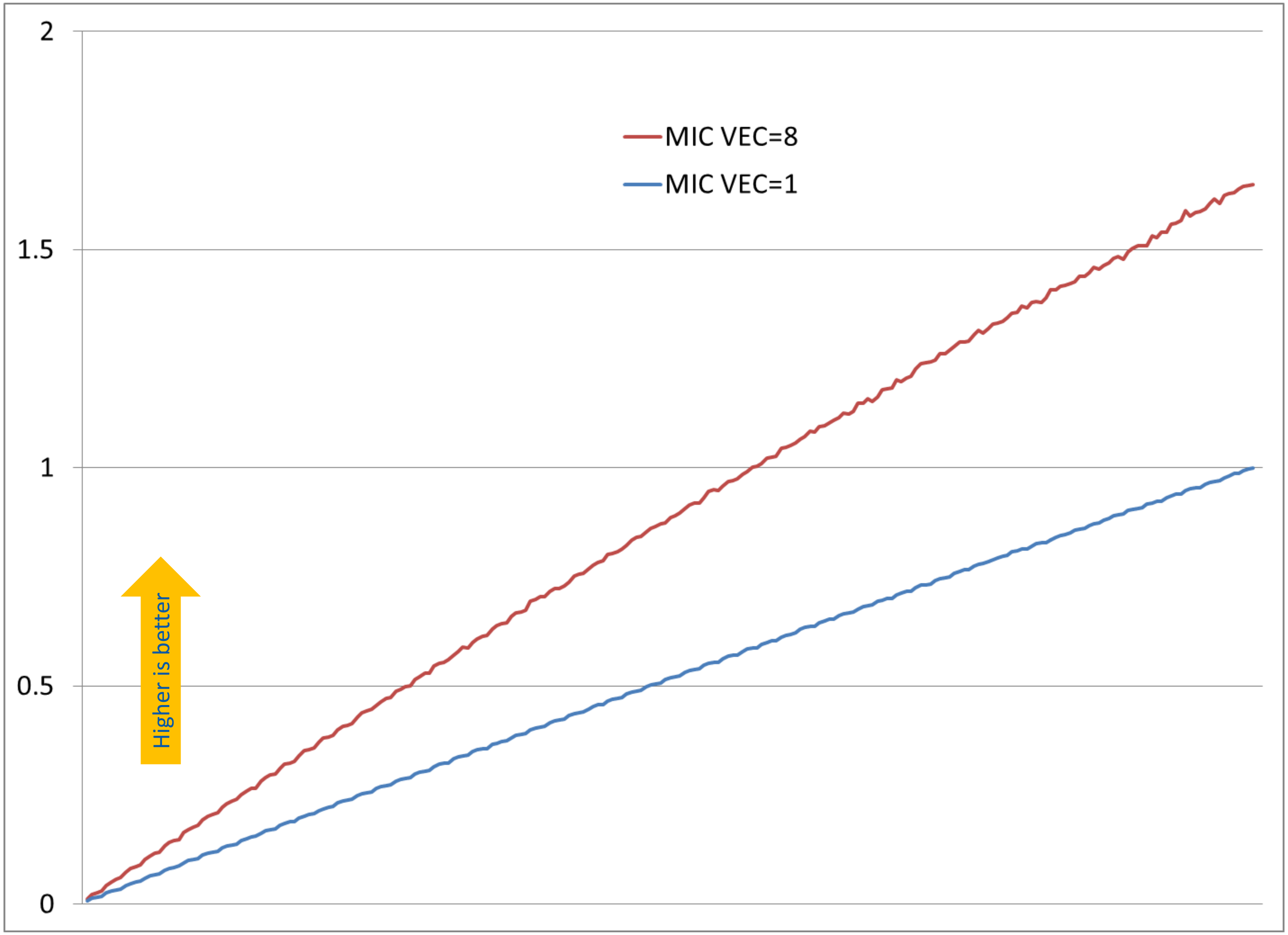
Reactive Atmospheric Chemistry Solver

- MIC implementation
  - Collapse and OpenMP thread of triply nested loop over cells of WRF-Chem domain
  - Dynamic thread scheduling
  - Add an inner dimension over cells for each array in Rosenbrock solver with masking to turn off cells as they reach convergence

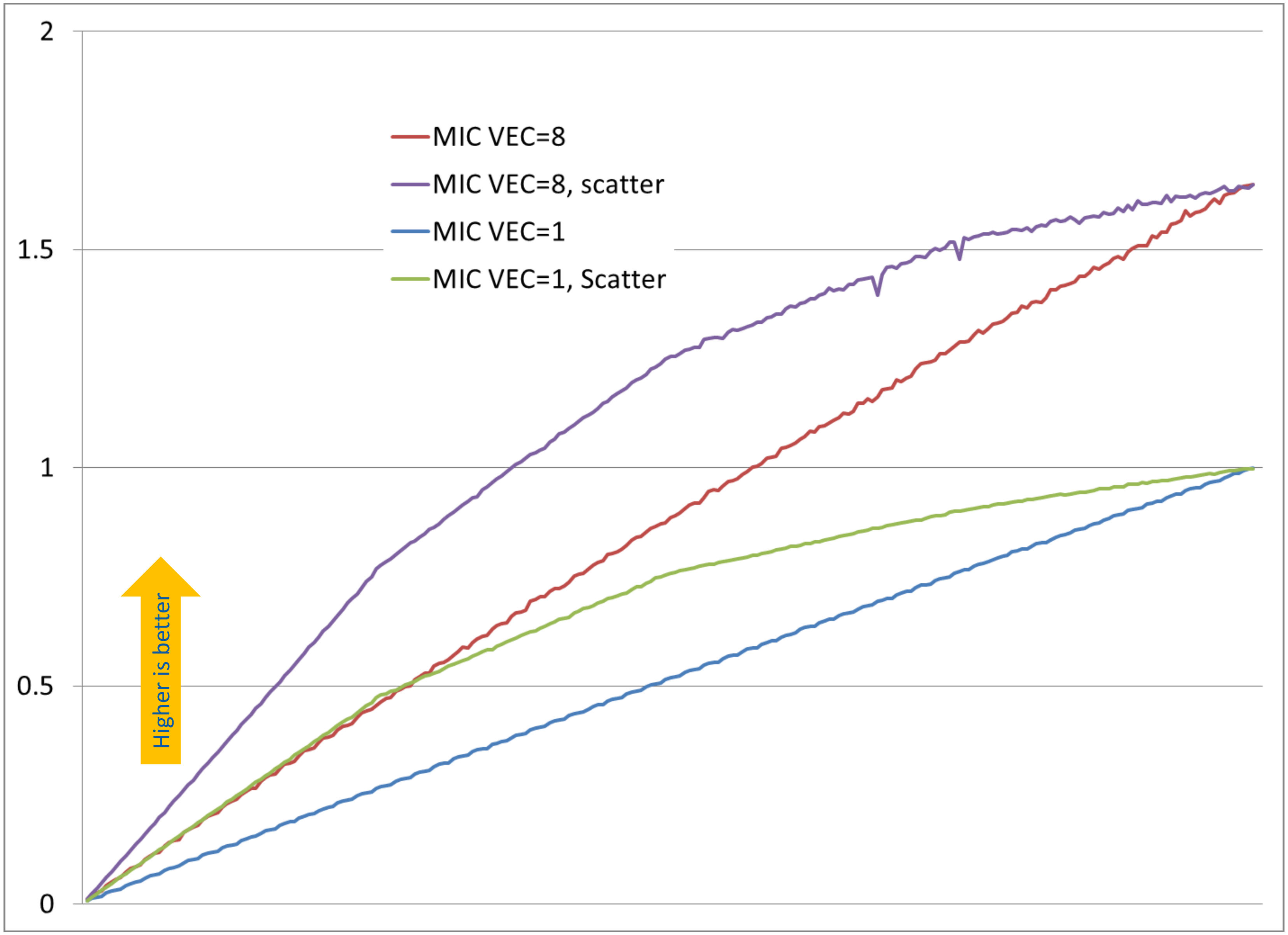
```
W( 53 ) = -a
W( 54 ) = W( 54 ) + a*JVS( 506 )
W( 55 ) = W( 55 ) + a*JVS( 507 )
W( 56 ) = W( 56 ) + a*JVS( 508 )
W( 57 ) = W( 57 ) + a*JVS( 509 )
W( 58 ) = W( 58 ) + a*JVS( 510 )
W( 59 ) = W( 59 ) + a*JVS( 511 )
```

```
WHERE ( vecMask )
  W0(:,53) = -a0(:)
  W0(:,54) = W0(:,54) + a0(:)*JVS0(:,506)
  W0(:,55) = W0(:,55) + a0(:)*JVS0(:,507)
  W0(:,56) = W0(:,56) + a0(:)*JVS0(:,508)
  W0(:,57) = W0(:,57) + a0(:)*JVS0(:,509)
  W0(:,58) = W0(:,58) + a0(:)*JVS0(:,510)
  W0(:,59) = W0(:,59) + a0(:)*JVS0(:,511)
```

Reactive Atmospheric Chemistry Solver



Reactive Atmospheric Chemistry Solver



Ongoing/Future work

- Continue investigating code transformations/ reformulations that will enhance locality and concurrency
- Adapt other atmospheric model kernels to Intel MIC
  - Tracer advection (similar to CAM-SE work)
  - Other physics, with fusion
- Whole code performance analysis and optimization
- Apply to other NREL mission-critical codes (OpenFOAM)

Summary

- Atmospheric models have abundant parallelism that must be fully utilized on extreme scale systems
- Many-/multi-core processors have potential to recover fine-grained and thread parallelism cost effectively (in both dollars and watts)
- Challenges: large memory footprint, low computational intensity, and flat profiles, performance programming